# THE USE OF FILE TIMESTAMPS IN DIGITAL FORENSICS

**Renico Koen[1], Martin S. Olivier[2]**

**[1]ICSA**
**University of Pretoria**
**South-Africa**
**[2]ICSA**
**University of Pretoria**
**South-Africa**

**[1]renico.koen@gmail.com, [2]martin@mo.co.za**

## ABSTRACT

Digital evidence is not well perceived by the human senses. Crucial pieces of digital evidence may simply be missed by investigators as the forensic significance of seemingly unimportant pieces of collected data may not be fully understood. This paper will discuss how abstract pieces of information may be extracted from seemingly insignificant evidence sources such a file timestamps by making use of correlating evidence sources. The use of file timestamps as a substitute for missing or corrupt log files as well as the information deficiency problem surrounding the use of timestamps will be discussed in detail. A prototype was developed to help investigators to determine the course of event as they occurred according to file timestamps. The prototype results that were obtained as well as prototype flaws will also be addressed.

## KEY WORDS

Digital Forensics, Event Reconstruction, Reco Platform, Timestamps.

# THE USE OF FILE TIMESTAMPS IN DIGITAL FORENSICS

## 1 INTRODUCTION

Digital evidence is not well perceived by the human senses [10]. Crucial pieces of digital evidence may simply be missed due to the fact that examiners do not fully comprehend how seemingly useless pieces of data can be converted to evidence of high value. This situation may be very problematic for digital investigators as it may help to create an incomplete picture of digital crimes under inspection [2]. It is therefore extremely important to examine all evidence, no matter how insignificant it may seem.

If an investigation team can understand an intruder's modus operandi, it may be possible to determine various attributes describing the intruder, such as skill level, knowledge and location [3]. Security mechanisms such as log files will usually be used to determine the actions of the intruder. Unfortunately it is possible that active security systems on the compromised system may be configured incorrectly or disabled completely [9]. In such circumstances investigators will have to turn to alternative sources of digital evidence.

File timestamps may serve as a worthy alternative, as timestamp information may be viewed as a simplistic log of events as they occurred. Although file timestamp information may be considered one-dimensional in a sense that it only records the time of the very last action that was performed on a file, it may still be a valuable source of evidence when very few alternatives remain. Unfortunately the processing of file timestamp information may be complicated by the sheer volume of available timestamps that should be processed.

The overabundance of digital evidence that need to be processed in small amounts of time could be described as an audit reduction problem [4]. The audit reduction problem describes the situation in which the presence of too much information obscures the focus point of investigations. Audit reduction would therefore be prevalent in digital evidence analysis due to the masses of files that needs to be inspected, spurred on by massive storage capacities of modern storage devices.

File timestamps analysis is an excellent example of the audit reduction problem: modern hard drives storage capacity may be anywhere in between the gigabyte to terabyte ranges; a very large number of files may be found on these devices — each file having different timestamp information associated with it. Although most of the file timestamps would be irrelevant to a case, a few may still be the key to its successful resolution. If these timestamps are simply overlooked, an incorrect conclusion could potentially be reached which may have dire consequences in store for the accused as well as the investigation team.

This paper will discuss the use of timestamps as a supplement or alternative to log files when log files are not available. The information deficiency problem, which describes the situation in which not enough information is available to allow investigators to get a clear picture of forensic significant events, will be discussed. This is done to inform the user of possible problems that may be experienced with alternative evidence sources. The concept of synergy applied to digital data is proposed as a solution to the information deficiency problem. The principle should allow investigators to use various insignificant evidence sources to generate abstract forms of information that are considered to be of forensic value. The paper is structured as follows. Section 2 will discuss the importance of file timestamps. Section 3 will focus on file timestamps related to incident phases. Section 4 will introduce the information deficiency problem and section 5 will discuss a possible solution to the problem. Section 6 will discuss the development of a prototype, section 7 will discuss the results obtained and section 8 will discuss the prototype flaws. Finally, section 9 will describe future work and section 10 will discuss the conclusion.

## 2 FILE TIMESTAMPS AS A SOURCE OF EVIDENCE

Attackers may try to delete or alter log files in an attempt to cover their tracks; fortunately pieces of information may still remain due to a lack of skills or access rights [9]. As an example, consider the use of well-known UNIX commands such as cat and grep. The attacker may use these two commands to remove identifying information from a system log file. A clever attacker may even change the log file's modification date after the alteration as not to arouse any suspicion from the system administrator. With the system log files compromised, investigators will have to find an alternative source of evidence as compromised evidence sources may not be credible in a court of law.

Fortunately there exists a less obvious source of digital evidence — file timestamps. Consider the example mentioned previously: the attacker used a combination of well-known tools such as the cat and grep commands to remove identifying information from the system log file. Very few attackers would actually reset the file access timestamps that were created when the shell command was executed. Even if they did manage to modify the file access times, they would have used a tool to do so. This means that although the commands used by the attacker do not have valid timestamps associated with it, a valid timestamp would be left somewhere on the system by the attacker, unless the command was executed from a read-only medium.

From the discussion it should be obvious that only extremely skilled attackers would be able to access a system without leaving a single trace; less skilled attackers are bound to leave small pieces of evidence behind that may ultimately be used to identify the responsible parties.

Popular file systems such as FAT, NTFS and EXT store file timestamps to keep record of:

- The file creation time
- Last time the file was accessed
- The last time the file was modified

These timestamps are updated by the underlying operating system when appropriate, but skilfully written applications also have the ability to manipulate timestamps as they require. Applications have different approaches concerning the management of timestamps. As an example, consider two well-known UNIX applications, namely cp and tar. When a file is copied using the cp command, the resulting creation and modification timestamps of the destination file would indicate the time that the cp command was executed. This is not the case with the tar command. When a compressed archive is created, the relevant files, along with their timestamps, are stored in a compressed archive. It should therefore be noted that some applications will posses timestamp modification capabilities which may have a negative effect on the timestamp analysis process. This topic will be discussed further in section 8.

## 3 TIMESTAMPS AND INCIDENT PHASES

Three digital evidence stages have been identified by Koen and Olivier [6] which classify evidence according to its temporal relationship with a digital incident. The identified stages are as follows:

- Pre-incident
- Incident
- Post-incident

The pre-incident stage focuses primarily on forensic readiness. Forensic readiness describes the extent to which a system is able to supply forensically-sound information to aid the digital investigation process [7]. Special software and hardware can be installed to monitor user actions and minimize the likelihood that the users of these systems can participate in mischievous activities without being noticed through policy management and the enforcement of restrictions. Suspicious activities may be captured and logged as required. The incident stage is concerned with the capture of digital evidence while a crime is being committed. The incident stage is primarily responsible for the capture and archiving of events as they occur in real time. The last stage is the post-incident stage in which the entire suspect and/or victim system's state is captured and analyzed after the digital crime has been committed. The phase is characterized by the mass-archiving of the states of the systems involved in the digital crime in an attempt to determine how the systems were used and by whom.

The information supplied by timestamps is very limited in a sense that a timestamp only records the last time a specific activity took place. To simplify this discussion, it will be assumed that a file will only have a single timestamp associated with it. Although this is not the case in reality, the principle will stay the same for timestamp-based information.

The most accurate timestamp from an evidence timeline classification point-of-view would be the timestamp recorded in the pre-incident stage as a timestamp with a time earlier than the incident means that the file in question was used before the incident occurred. This means the file may have executed an action on files involved with the incident, but it could only have done so up until the point that it was last loaded in memory. Timestamps captured in the incident stage indicate that the files in question were used during the incident stage, but could also have been used during the pre-incident stage. The situation gets worse in the post-incident stage: files with timestamp in this stage may have had actions performed on them during any one of the phases. An information deficiency problem therefore exists with regards to timestamps and the incident stage and especially the post-incident stage.

For analysis purposes it will have to be assumed that evidence had actions performed on it in every stage prior to its current incident stage. A solution to the information deficiency problem may be to introduce additional evidence sources in an attempt to build a timelines that indicate upper and lower bound incident stages in which actions were performed on the object in question.

## 4 APPLICATIONS AND FILE TIMESTAMP RELATIONSHIPS

In order for a timestamp to change an action is needed. The action will have to be triggered by an application or device driver resident in memory at the time of change. For this discussion it is assumed that three types of timestamps exist, namely the creation, modification and access timestamp and that the operating system alone can modify file timestamp values. The value of the timestamp is not important in this example as its meaning is largely dependent on the application that triggered the event. What should be considered important is the fact that an executable code that triggered the event to be executed should have been active in physical memory prior to triggering the event. This means that the file in question should have been loaded into memory, thus modifying its file access timestamp. An executable that accesses or modifies a file should therefore have an file access timestamp which is smaller than the file in question's timestamp (create, read or modify depending on the action performed). The following macro can be defined to determine if an application's create, access or modify time has been edited:

$$\textit{touched(f)} = \textit{ceil ( create(f), access(f), modify(f) )}$$

Using the defined macro, the following condition should therefore hold:

$$\textit{access(executable)} <= \textit{touched(file)}$$

Unfortunately due to the information deficiency problem identified previously, a piece of executable code may be loaded again in the future which means that the stated condition will not hold anymore as the access time of the executable code changed. The following situation may therefore exist:

*access (executable) <= touched(file) or access (executable) >= touched(file)*

This basically means that it would not be possible to pinpoint the application responsible for the modification of a file as not enough information exists. If the timestamp found on an executable piece of code shows that the executable was last accessed before a file timestamp was last modified it does not necessarily rule out the executable as the accessory or modifier of the file in question as some sections of code may stay resident in memory for a period of time before it actually accessed the file. It can therefore be concluded that application/file timestamp relationships is of very little forensic significance on its own; some additional form of information is needed to help to rule out executables that could not have modified the file in question. The executable access timestamp cannot be used to help rule out the application associated with it as the application may have been resident in memory for some time before it triggered the modification of a file's timestamps. If it were possible to prove that the application in question was removed from memory some time after its file access timestamp indicated, it may be enough to rule out the application as the trigger source.

As an example, consider the diagram illustrating the executable access timestamps in the different incident stages (see figure 1). Various evidence artefacts have been organized according to file creation timestamp dates. As discussed previously, application/file timestamp relationships are not of forensic significance on their own; executable 1, 2 and 3 could therefore individually have created files A, B, C, D and E. It is therefore not possible to rule out any executables from the equation.

Imagine the intruder managed to reboot the system in question during the incident phase. Knowledge of this event may help to place an upper-bound on the last possible time that executable 1 could have had an effect on the file timestamps of the listed artefacts. File access information informs us that executable 1 was last executed during the pre-incident phase; a system log file (collaborating evidence) shows us that the system went offline during the incident phase. Executable 1 was not loaded again after the system went back online after the reboot. It can therefore be concluded that executable

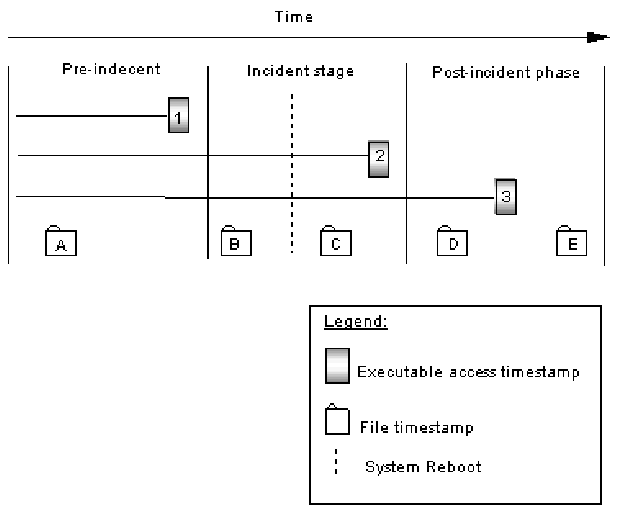


*Figure 1: Files organized according to timestamp information.*

1 did not have an effect on the timestamps of the listed artefacts after the reboot. With enough collaborative evidence at hand it may be possible to narrow the list of possible executables down substantially which may have been responsible for triggering an event that modified timestamp information. This example relied on the knowledge that a system rebooted. Normally such information will be gathered from a system log file, but in the absence of credible log files, investigators may once again need to turn to file access timestamps as an indicator of system events. When a system boots, various executables are loaded as services. These executables are usually only loaded once and stay loaded until a system halts or reboots. By looking at the access timestamps of these services it may actually be possible to determine when the system booted. This method will be discussed further in section 6.

## 5 SOLUTION TO THE INFORMATION DEFICIENCY PROBLEM

Synergy describes the situation in which the whole is greater than the sum of the parts [11]. Although the discussed events may be seen as insignificant on their own, their importance may increase when their collective importance is realized, therefore when a state of synergy is achieved.

Consider the example in figure 1 again: each of the events that caused changes in timestamp information associated with the files is of very little forensic value when considered on their own. Even the timestamp that indicated that the system in question performed a boot operation would seem relatively useless on its own as it does not convey any useful information other than a system boot took place. The real value in the timestamp information lies in the fact it represents events that took place. On a higher level these events may be related with one another to create an abstract view of the events as they occurred.

The example in the previous section illustrates that it may be possible to extract useful information from seemingly useless data when viewed on its own. A file's access timestamp may have very little importance on its own; its importance is directly related to the importance of the event that it represents. A principle based on synergy that focuses on the creation of abstract evidence information from insignificant pieces of data may therefore be formulated as follows:

*Event data is generated when a significant digital event occurs. Although the generated event data is of little value when viewed independently, collectively event data can produce information that can help investigators to deduce relationships between events to produce abstract views of the evidence at hand.*

Investigators usually have lots of complex questions to answer in a short period of time [3]; the possibility therefore exists that evidence may be overlooked as investigators focus their attention to evidence that seems more important in an attempt to save valuable time. Identifying the relationships that may exist between seemingly unimportant pieces of digital evidence may be an extremely tedious task to perform. As Adelstein [1] points out, it is not feasible for investigators to manually analyze storage devices with storage capacities in access of gigabytes as there is just too much data to process. Without some form of automated processing the benefit obtained as a result of time invested by investigators would be minimal due to the sheer volumes of data that needs to be processed.

## 6 PROTOTYPING

A prototype has been created based on application/timestamp relationships discussed previously in an attempt to illustrate the defined principle in action. The prototype was developed to extract information from Linux-based EXT2/3 file system storing ordinary files, applications and system logs. The prototype was built under the assumption that the file timestamps have not been tampered with. It has also been assumed that the executable access time indicated the last time the application was loaded by the operating system. File creation timestamps were ignored as it is assumed that file access and modification times will always be larger than a file's creation time.

Casey [2] proposed a certainty scale that may be used to determine the level of trust that can be placed in the information deduced by the investigators by examining the forensic evidence at hand. Evidence that appears highly questionable will have a low certainty level associated with it while evidence that can be correlated with other captured evidence sources will receive a higher certainty rating. Casey's certainty scale can be used in addition to the defined principle to increase the level of trust experienced with extracted information; evidence which can be correlated with other sources of information may experience a higher degree of certainty.

Relating Casey's work and the defined principle to timestamp information it can be assumed that timestamp information that is correlated with timestamp information from the same disk image will have a lesser degree of certainty than timestamp information that may be related to some other form of evidence, such as system logs. The prototype was built with the purpose of identifying the last possible time that an application could have been loaded in memory, known as the last possible execution time. This was done in an attempt to determine which files could have been modified by the application in question. The last possible execution time is determined in one of two ways: by correlating an application's access timestamp with system log entries or by correlating an application's access timestamp with the access timestamps of system applications and/or files that are accessed on system boot or shutdown events.

The first method would obviously be the better choice for the correlation of evidence as it contains a rich source of system-related history information. To determine the last possible time an application could have been in memory is simple: use the application in question's access timestamp and search for the earliest system halt or reboot event that occurred after the access time in the log file. The time specified in the log for the halt or reboot event would therefore serve as the last possible execution time as the executable was never accessed again after that specific point in time. The second method may serve as an alternative to log files in situations when it has become evident that the system log files have been tampered with or in environments where no log files exist. When an operating system boots or halts, it will load various system applications and access stored settings, changing their accessed timestamps. The timestamps viewed on their own are insignificant, but when used to determine when a system was turned on or off, it may be of great value to forensic investigators. As an example, consider the sequence of events that occurs when a standard Linux system boots. The first process created by the kernel executes the /sbin/init application. When the /sbin/init application starts, it reads the /etc/inittab file for further instructions. By simply checking the accessed timestamps of either one of the two files it would be possible to determine the last time that a system booted. It can be argued that the information is also obtainable from alternative sources (such as the /proc/uptime file), but in situations where the alternative is damaged or simply does not exist, timestamps will have to suffice. Calculating the last possible execution time for the second technique is similar to the method used to determine the last possible execution time for the first method: determine an application's accessed timestamp information and determine the last time a system booted or halted by looking at the applications and files associated with the system boot or halt operations.

The prototype reads disk images to produce XML files containing timestamp information. These XML files are then converted to scatter charts to improve the way timestamp information is perceived by the human senses. The prototype depends on two freely available libraries, namely the Reco Platform [5] and JFreeChart [8]. The design is illustrated in figure 2.

The Reco Platform supplies low-level EXT2/3 support to the system while the JFreechart library supplies the graphing functionality required by the application. The prototype source code has been released under the GNU GPL license and is available on Sourceforge [5]. The next section will discuss the results that were obtained using the developed prototype in more detail.

## 7 RESULTS

The prototype was tested using Linux (Fedora Core 4). A disk image was made and last possible execution times were computed for each application

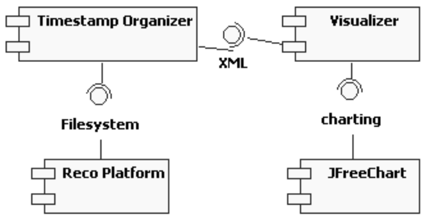
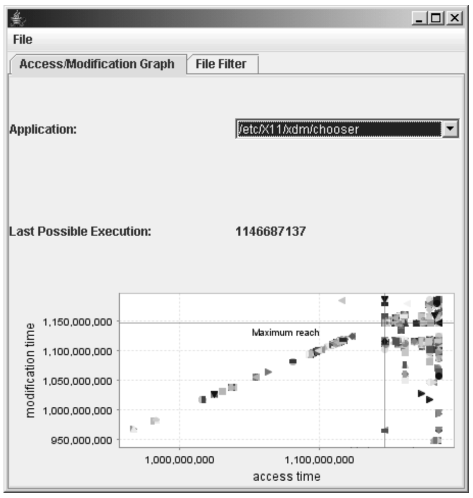


*Figure 2: The prototype design.*



Application: /etc/X11/xdm/chooser

Last Possible Execution: 1146687137

*Figure 3: A screenshot of the prototype.*

using both methods described previously to produce separate XML files. A scatter chart was constructed using each detected file's modification and access times as coordinate values. A selected application's last possible execution time was plotted as horizontal and vertical lines to indicate the reach (in terms of what the application could have modified) of the application in question. Figure 3 illustrates the produced scatter chart as well as the horizontal and vertical lines indicating the maximum reach of the application in question.

The user is allowed to select an application of interest in a dropdown control populated with a list of applications. The application's last possible execution time is computed and plotted on the scatter chart upon selection.The last possible execution time, access time and modification times are represented by an integer value; the integer value is a timestamp that describes the amount of seconds that have elapsed since January 1,1970 (which means that the values could easily be manipulated using a function such as ctime) when the event in question occurred or should occur. In the example (figure 3) the last possible execution time for the application */etc/X11/xdm/chooser* was calculated to be 1146687137 seconds since 1 January 1970. Translated to human-understandable terms, the last possible execution time for the application in question is Tuesday, May 2, 2006 at 23:58:57. The application cannot be responsible for any file access or modification operations performed after the last possible execution time, represented by the horizontal and vertical lines on the graph. Any files outside of the horizontal and vertical lines will therefore have been accessed or modified by other applications.

By simply looking at the generated chart it is possible to visually detect which files could have been modified by the application in question. Due to the sheer magnitude of the amount of files that are stored on a disk drive, a file filter functionality has been added to the prototype to search for files with timestamps conforming to specific criteria. Determining the names of the files that could have been modified by the application in question was as simple as submitting a filter query that contained the last possible execution time of the application in question.

A comparison between the two techniques used to determine an application's last possible execution time yielded the results that were expected: since system log files contain detailed history information, more accurate last possible execution times could be calculated leading to more accurate results. File access timestamps contain only the last time the file was accessed and can therefore be compared to a log file containing entries which date back to the last time a system in question was booted. This implies that the method could work with the same efficiency as the first method in a scenario where a system rarely goes offline. However, this method would be very inaccurate for systems that goes offline frequently.

## 8 CRITICISMS

As discussed in section 2, some applications have the ability to modify timestamps. The work in this paper assumed that the timestamps are modified by the operating system only and did not take into account that applications may manipulate the proposed analysis method by changing file timestamps to render the method invalid. In reality, interpreted meaning of a timestamp is therefore largely dependent on the way in which the application responsible for the creation or modification of a file manages timestamp information.

It has also been assumed that applications will be stored on a writable medium; an application's timestamp information will therefore be updated each time the application is loaded into memory. This may not necessarily be the case as it is possible in UNIX environment to mount file systems in read-only mode. This means that an application's file access time will not change rendering the method described in this paper useless.

Another concern is that an application may have accessed or modified a suspicious file prior to its last possible time of execution; if the suspicious file was accessed or modified again some time later in the future (presumably after the application in question's last possible time of execution), the timestamp may be labelled as being out of reach of the application in question. Technically this is true as the file was last modified by another application, but this situation may not always be desirable. A way to overcome this problem is to divide application timestamps into the various incident stages discussed in section 3. Only applications with access timestamps falling in the incident and post-incident phase will have to be considered for inspection as it can be assumed that applications with last possible execution times falling in the pre-incident stage were not involved with the incident in question.

## 9 FUTURE WORK

A complex application would typically touch various files while it is executing. A typical scenario would be where the application in question first accesses its configuration files and then data files. By describing an application's actions formally, it may be possible to create a profile that accurately describes an application's file access characteristics.

Another topic that requires attention is the inspection of the file access of an operating system's boot process. When an operating system performs the boot process, various files will be accessed. Different operating systems would access different files which creates the possibility that the file access operations performed by an operating system could potentially be used as a fingerprint to help operating system identification in circumstances in which conventional methods are not deemed appropriate. The described process could potentially be improved by adding the concept of a termination signature. The termination signature describes the characteristics of an application when it terminates, in other words what actions it takes just before it terminates. If such a signature can be incorporated into the concepts described in this paper, more accurate results may be obtained.

## 10 CONCLUSION

This paper discussed how timestamps could be used to rule out files that could not have been modified by distinct applications based on an application's calculated last possible execution time. A principle was introduced based on the concept of synergy claiming that insignificant pieces of event datum may collectively be of significant forensic importance. A prototype was constructed based on this principle, using timestamps as a source of insignificant evidence. The prototype calculated various applications' last possible execution times and visually depicted the information in a manner that can easilly be understood by the observer. The prototype helped to visualize abstract digital data which are not well-perceived by the human senses to help investigators to easily understand the produced data as well as its importance. Unfortunately the method used by the prototype is not absolute in a sense that it cannot successfully be applied to all environments under all conditions. It has become evident that a great need exists for ways in which digital evidence can be visualized. More research will have to be conducted to find ways to visualize digital information to allow investigators to easily understand digital evidence at hand.

## References

[1] Adelstein, F. Live forensics: diagnosing your system without killing it first. *Commun. ACM 49*, 2 (2006), 63–66.

[2] Casey, E. Uncertainty, and loss in digital evidence. *International Journal of Digital Evidence 1*, 2 (2002).

[3] Casey, E. Investigating sophisticated security breaches. *Commun. ACM 49*, 2 (2006), 48–55.

[4] Corey, V., Peterman, C., Shearin, S., Greenberg, M. S., and Bokkelen, J. V. Network forensics analysis. *IEEE Internet Computing 6*, 6 (2002), 60–66.

[5] Koen, R. Reco platform homepage. Online: http://sourceforge.net/projects/reco, June 2007.

[6] Koen, R., and Olivier, M. An open-source forensics platform. In *SAICSIT '07. Proceedings of the Annual SAICSIT conference* (2007).

[7] Mohay, G. Technical challenges and directions for digital forensics. In *SADFE '05: Proceedings of the First International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'05) on Systematic Approaches to Digital Forensic Engineering* (Washington, DC, USA, 2005), IEEE Computer Society, p. 155.

[8] ORL. JFreechart. Online: http://www.jfree.org/jfreechart, Online: July 2007.

[9] Stallard, T., and Levitt, K. Automated analysis for digital forensic science: Semantic integrity checking. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference* (Washington, DC, USA, 2003), IEEE Computer Society, p. 160.

[10] Wang, S.-J. Measures of retaining digital evidence to prosecute computer-based cyber-crimes. *Comput. Stand. Interfaces 29*, 2 (2007), 216–223.

[11] Wikipedia. Synergy. Online: http://en.wikipedia.org/w/index.php?title=Synergy, July 2007.